

**LA-UR-21-23851**

Approved for public release; distribution is unlimited.

Title: Modeling the Fission Meter

Author(s): Marcath, Matthew James  
Mayo, Douglas R.  
Bates, Cameron Russell

Intended for: Report

Issued: 2021-04-20

---

**Disclaimer:**

Los Alamos National Laboratory, an affirmative action/equal opportunity employer, is operated by Triad National Security, LLC for the National Nuclear Security Administration of U.S. Department of Energy under contract 89233218CNA000001. By approving this article, the publisher recognizes that the U.S. Government retains nonexclusive, royalty-free license to publish or reproduce the published form of this contribution, or to allow others to do so, for U.S. Government purposes. Los Alamos National Laboratory requests that the publisher identify this article as work performed under the auspices of the U.S. Department of Energy. Los Alamos National Laboratory strongly supports academic freedom and a researcher's right to publish; as an institution, however, the Laboratory does not endorse the viewpoint of a publication or guarantee its technical correctness.

# Modeling the Fission Meter

Matthew J. Marcath, Douglas R. Mayo, and Cameron R. Bates

31 March 2021

# Contents

0.1	Introduction . . . . .	2
0.2	Python script: ptrac_feynman_y.py . . . . .	14
0.3	MCNP6 model: Cf-252 point source . . . . .	26
0.4	Python script configuration: Cf-252 point source . . . . .	32
0.5	MCNP6 model: TAI 1 . . . . .	34
0.6	Python script configuration: TAI 1 . . . . .	40
0.7	Python results plotter . . . . .	42

## **0.1 Introduction**

This presentation includes an overview of the process, example inputs, and example results for the Fission Meter and its response.



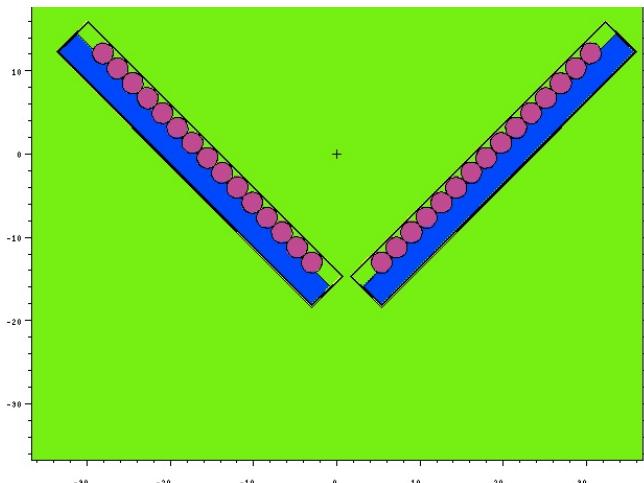
# Modeling the Fission Meter

Matthew J. Marcath, Douglas R. Mayo, and  
Cameron R. Bates

6 April 2021

# Fission Meter: MCNP model and Feynman-Y

- Fission Meter is modeled in MCNP.
- MCNP model produces a PTRAC file of He-3 captures.
- Python script reads the PTRAC file and produces Feynman-Y results.
- Software you need:
  - MCNP6 – preferably LANL’s development version with HDF5.
  - Python 3.7 or greater with SciPy
  - MCNPTools – preferably 5.2.0 for HDF5
- All software is or can be made available on LANL HPC

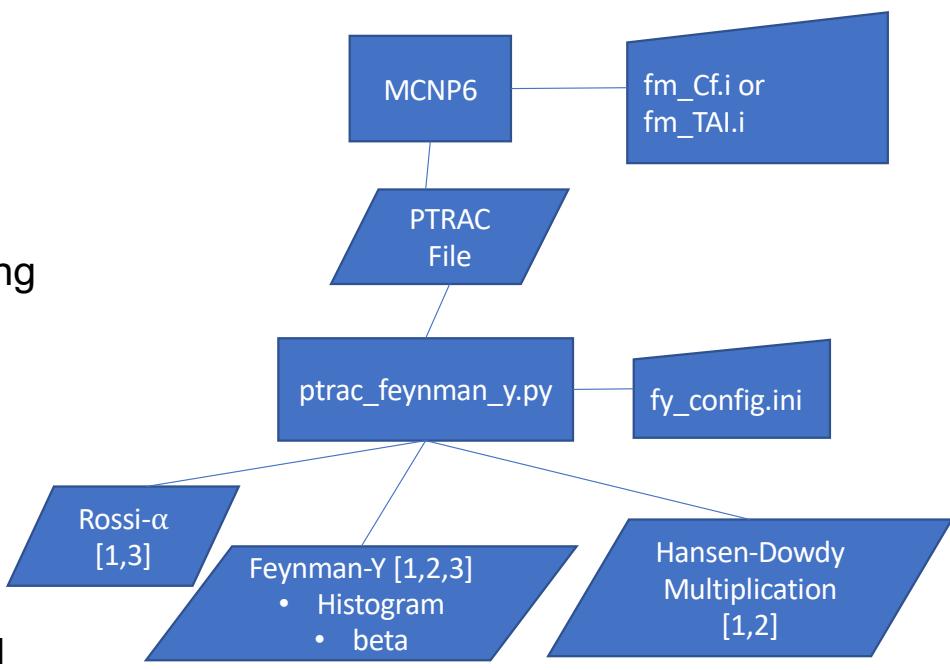


- Example contains:
  - fm\_Cf.i and fm\_TAI.i - MCNP input
  - ptrac\_feynman\_y.py - analysis script
  - fy\_config.ini – analysis script configuration file
  - plot\_n\_coinc.py - results plotter
  - Outputs



## Work flow

- MCNP writes a ptrac file containing events with He-3 captures in the tube's active volumes.
- Python script
  - Time distributes MCNP histories
  - Applies dead time.
  - Makes Rossi- $\alpha$  histogram and die-away fit.
  - Makes Feynman-Y histograms and Hansen-Dowdy multiplication estimates.



1. Smith-Nelson, M., et al., "Neutron Specialist Handbook and Informational Text (Unclassified Version)", LA-UR-07-6170, 2007.
2. Dowdy, E J, Hansen, G E, & Robba, A A. "Feynmann variance-to-mean method." United States, 1985.
3. Ensslin, N., "Passive Nondestructive Assay of Nuclear Materials: Principles of Neutron Coincidence Counting." LA-UR-90-732, 1991.



# Analysis script config: Cf-252 point source

```
# General script parameters
[general]
ptrac_file = fm_cf_mcnp.1e8.out.p.h5
output_ext = 1e8.out
# History activity in s^-1
hist_rate = 1e5
# True or False: read ptrac, time distribute, apply dead time, write
create_event_list = True
# Event list path to read; if you already have one and want to skip
#           to multiplicity analysis.
event_list_path = event_list_td_dt.1e8.test.npy
multiplicity_analysis = True

# system and material parameters
[system]
# Uncorrelated neutron rate in [n/s] i.e. (alpha,n)
S_0 = 0
# Rossi alpha histogram range [us]
rossi_max_time = 100
rossi_bin_step = 1
rossi_accidental_start = 75
rossi_fit_start = 5
rossi_fit_end = 50

# Feynman-Y gate widths [s]
sweep_start = 1e-6
sweep_end = 256e-6
sweep_step = 4e-6

# Detector dead time [s]
tau = 3.125e-08

# Starter fission rate in [f/s]
F_0 = 3.75e5

# Total neutron source strength [n/s]
NSS = 376727.51

# nu_bar in starter material [n/f]
nu_bar_0 = 3.76

# nu_bar in multiplying material [n/f]
nu_bar_1 = 4.415

# Diven-number for starter material
D_0 = 3.23

# Diven-number for multiplying material
D_1 = 3.23

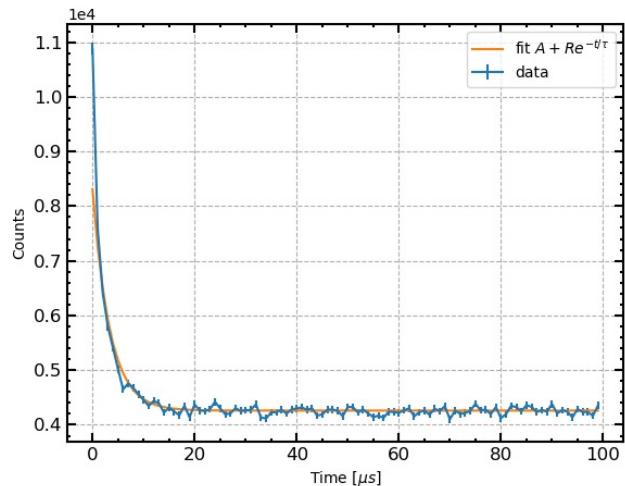
# Transmission ratio
psi = 1
```

Parameters follow  
Hansen-Dowdy  
formulism.



## Results: Cf-252 point source

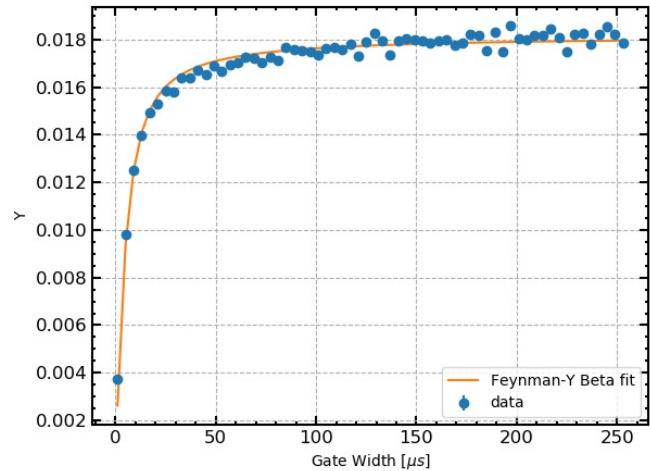
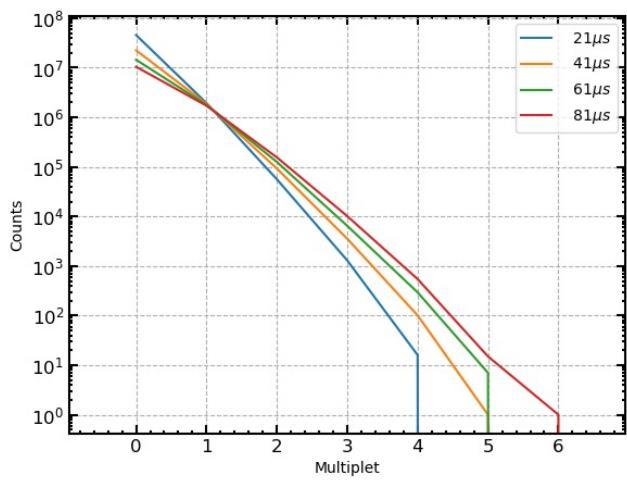
- Results and fits are printed to text file.
- Results saved to NumPy files
  - Event list
  - Rossi- $\alpha$
  - Feynman-Y multiplicity
  - Feynman-Y  $\beta$



A=4.26E+03 R=4.05E+03  $\tau$ =3.4 us



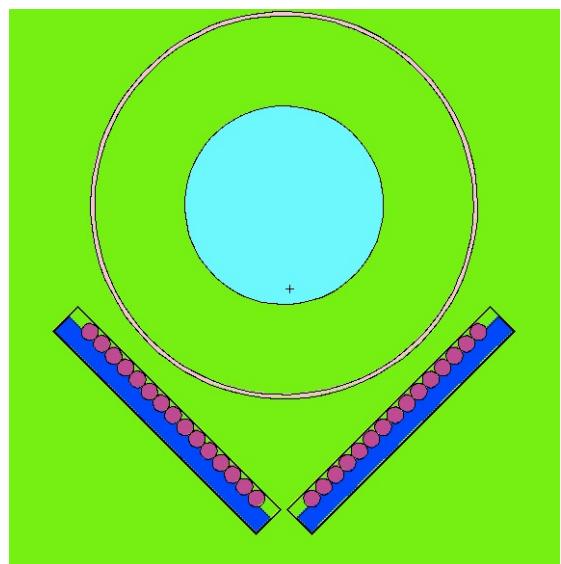
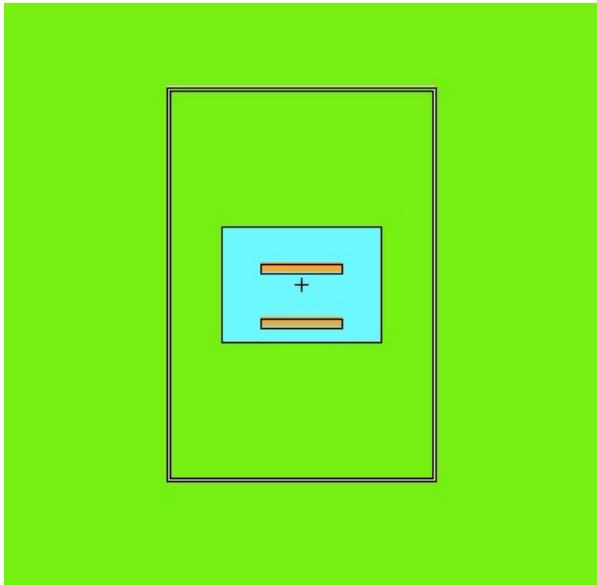
## Results: Cf-252 point source



$$K = 1.8E-02 \quad \beta = 3.4E+05 \text{ s}^{-1}$$



## MCNP Model: Type 1 TAI



# Analysis script config: Type 1 TAI

```
# General script parameters
[general]
ptrac_file = fm_tai_mcnp.1e8.out.p.h5
output_ext = 1e8
# History activity in s^-1
hist_rate = 6.45455e5
# True or False: read ptrac, time distribute, apply dead time, write
create_event_list = True
# Event list path to read; if you already have one and want to skip
#           to multiplicity analysis.
event_list_path = event_list_td_dt.1e8.npy
multiplicity_analysis = True

# system and material parameters
[system]
# Uncorrelated neutron rate in [n/s] i.e. (alpha,n)
S_0 = 0
# Rossi alpha histogram range [us]
rossi_max_time = 100
rossi_bin_step = 1
rossi_accidental_start = 75
rossi_fit_start = 5
rossi_fit_end = 50
```

```
# Feynman-Y gate widths [s]
sweep_start = 30e-6
sweep_end = 400e-6
sweep_step = 8e-6

# Detector dead time [s]
tau = 3.125e-08

# Starter fission rate in [f/s]
F_0 = 6.45455e5

# Total neutron source strength [n/s]
NSS = 2424860.

# nu_bar in starter material [n/f]
nu_bar_0 = 3.76

# nu_bar in multiplying material [n/f]
nu_bar_1 = 3.76

# Diven-number for starter material
D_0 = 3.23

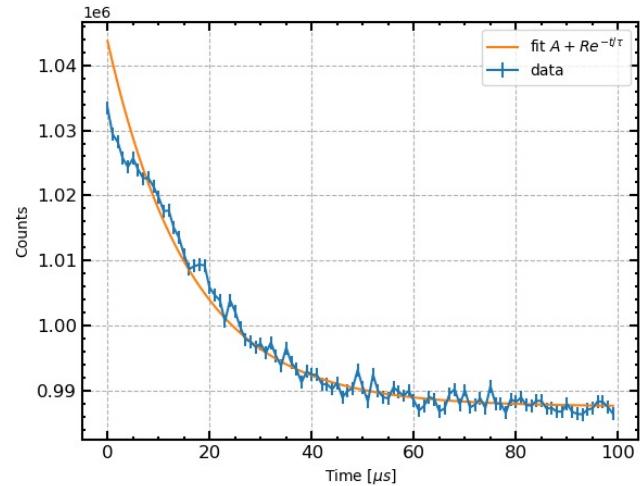
# Diven-number for multiplying material
D_1 = 3.23

# Transmission ratio
psi = 1.0
```

Parameters follow  
Hansen-Dowdy  
formulism.

## Results: Type 1 TAI

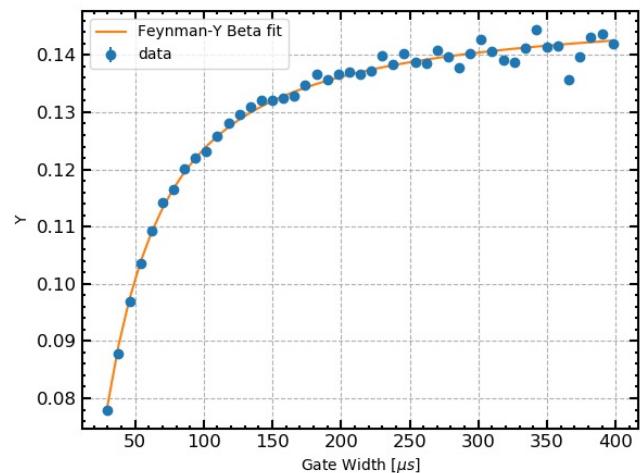
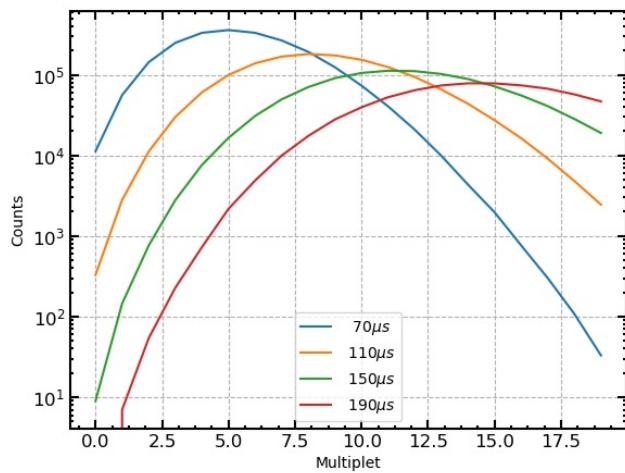
- Results and fits are printed to text file.
- Results saved to NumPy files
  - Event list
  - Rossi- $\alpha$
  - Feynman-Y multiplicity
  - Feynman-Y  $\beta$
- Bad multiplication estimate  $p_f \ll p_c$



$A = 9.88E+05 \quad R = 5.61E+04 \quad \tau = 16 \text{ us}$



## Results: Type 1 TAI



$$K = 1.49E-01 \quad \beta = 5.87E+04 \text{ s}^{-1}$$



NSA

Managed by Triad National Security, LLC., for the U.S. Department of Energy's NNSA.

3/31/21

11

## 0.2 Python script: ptrac\_feynman\_y.py

Given a configuration file, the python script processes MCNP PTRAC outputs to produce Rossi-alpha and Feynman-Y histograms.

```
1 #!/usr/bin/env python3
2 #
3 # 17 April 2020 Marcath
4 # rewrite Dec 2020
5 # load a ptrack file and look for He-3 captures for Feynman-Y
6 #    ↪ and
7 #    ↪ Rossi-alpha analysis. Uses Feynman-Y and Rossi data to
8 #    ↪ compute
9 #    ↪ total multiplication M_T.
10 from mcnp tools import Ptrac
11 import numpy as np
12 import matplotlib.pyplot as plt
13 from math import floor, exp
14 from scipy.optimize import curve_fit
15 import configparser
16 import sys
17
18 # FeynmanBetaFunc
19 # The function is used primarily for fitting.
20 def FeynmanBetaFunc(T_0, K, beta, tau):
21     return (np.multiply(np.divide(K*(T_0-tau), T_0),
22                         1-np.divide((1-np.exp(-beta*(T_0-tau))), (beta*(T_0-tau)))))
23
24 # FeynmanBetaFit
25 def FeynmanBetaFit(Y_m, gates, tau, fid):
26     fid.write('FeynmanBetaFit...\n')
27     fits, cov = curve_fit(FeynmanBetaFunc, gates, Y_m, bounds=(0,
28         ↪ [1e6, 1e9]))
29     fits, cov = curve_fit(lambda T_0, K, beta: FeynmanBetaFunc(T_0
30         ↪ , K, beta, tau), gates, Y_m, bounds=(0, [1e6, 1e9]))
31     fid.write('\tK: {0:e}\tbeta: {1:e}\n'.format(fits[0], fits[1]))
32
33 # FindMTYC calculates dead-time corrected M_T
34 # takes uncorrected Feynman-Y (Y_m, sigma_Y_m), window width (
35 #    ↪ T_0 in
```

```

35 #     seconds), mean counts in window (C_bar), inverse neutron
36 #     ↪ life
37 #     (beta in seconds), print file id (fid).
38 #     Uses Hansen–Dowdy formulism to iteratively solve for dead-time
39 #     corrected Feynman–Y and total multiplication (Y_c and M_T).
40 def FindMTYC(Y_m, sigma_Y_m, T_0, C_bar, beta, tau, S_0, F_0,
41   ↪ NSS, psi, nu_bar_0, nu_bar_1, D_0, D_1, fid):
42   T = T_0 - tau
43   epsilon = C_bar/T_0*(1/NSS)*psi
44   R = S_0/(nu_bar_0*F_0)
45   D_bar_0 = D_0/(1+R)
46   g = 1 - (1-exp(-beta*T))/(beta*T)
47   A = T/T_0*epsilon*g*exp(-beta*tau)
48   fid.write('tGivens: \n')
49   fid.write('t\ty_m: {0:f}\tC_bar: {1:f}\n'.format(Y_m,C_bar))
50   fid.write('t\tS_0: {0:f}\tF_0: {1:f}\tNSS: {2:f}\n'.format(
51     ↪ S_0, F_0, NSS))
52   fid.write('t\tbeta: {0:f}\tT_0: {1:f}\ttau: {2:f}\tPsi: {3:f}
53     ↪ }\n'.format(beta, T_0, tau, psi))
54   fid.write('t\tnu_bar_0: {0:f}\tnu_bar_1: {1:f}\tD_0: {2:f}\\
55     ↪ tD_1: {3:f}\n'.format(nu_bar_0,nu_bar_1,D_0,D_1))
56   fid.write('tCalculated: \n')
57   fid.write('t\tepsilon: {0:f}\tg: {1:f}\tA: {2:f}\n'.format(
58     ↪ epsilon,g,A))
59
60   # M_T(Y_c=Y_m), h(M_T), Y_c(h), M_T(Y_c) ... until small change
61   #     ↪ in M_T
62   Y_c_old = 1e-6
63   Y_c = Y_m
64   while (Y_c-Y_c_old)/Y_c_old > 0.05:
65     M_T = 0.5*(1 - D_bar_0/D_1 + ((1 - D_bar_0/D_1)**2 + 4*Y_c/(
66       ↪ A*D_1))**0.5)
67     h = C_bar/T_0 + 0.5*epsilon*beta*D_1*(M_T-1)
68     Y_c_old = Y_c
69     if tau > 0:
70       Y_c = (Y_c_old + 2*tau*C_bar/T_0)/(1-4*tau*h)
71     else:
72       break
73
74   sigma_M_T = sigma_Y_m/(A*D_1)*((1-D_bar_0/D_1)**2+4*Y_c/(A*D_1
75     ↪ ))**-0.5
76
77   fid.write('t\tM_T: {0:f}\tsigma_M_T: {1:f}\tY_c: {1:f}\n'.
78     ↪ format(M_T,sigma_M_T,Y_c))
79   return M_T, sigma_M_T, Y_c

```

```

70
71 # FitRossiA fits Rossi-A data
72 # Takes bins (us) and counts. Prints to file (fid).
73 # Fits using S(t)=A+R*e^{-t/tau} to return A, exp(R), -1/tau (us
74 #   ↪ ).  

74 def FitRossiA(bins,counts,accidental_start, fit_start, fit_end,
75 #   ↪ fid):
75 fid.write('FitRossiA ...\\n')
76 fid.write('\\tFitting data to S(t) = A + R*e^{-t/tau}.\\n')  

77
78 accidental_mask = [bins>accidental_start]
79 accidentals = np.sum(counts[accidental_mask])/np.sum(
80 #   ↪ accidental_mask))
80 fid.write('\\tAccidental range over last {0:d} bins.\\n'.format(
81 #   ↪ np.sum(accidental_mask)))
81 fid.write('\\tAccidental rate: {0:f} per bin \\n'.format(
82 #   ↪ accidentals))  

83  

83 #accidental_range = 50
84 #accidentals = np.sum(counts[np.size(bins)-accidental_range:]/
84 #   ↪ accidental_range)
85 #fid.write('\\tAccidental range over last {0:d} bins.\\n'.format(
85 #   ↪ (accidental_range)))
86 #fid.write('\\tAccidental rate: {0:f} per bin \\n'.format(
86 #   ↪ accidentals))  

87
88 fit_mask = [bins<fit_end]
89 if np.size(np.nonzero((counts[fit_mask]-accidentals)<0)) > 0:
90     if fit_end > np.nonzero((counts[fit_mask]-accidentals)<0)
91         [0][0]:
91         fit_end = np.nonzero((counts[fit_mask]-accidentals)<0)
91         [0][0]
92 fit_mask = [bins>fit_start] and [bins<fit_end]
93 fid.write('\\tFit over {0:3.2f} us to {1:3.2f} us bins.\\n'.
93 #   ↪ format(fit_start, fit_end))  

94
95 #fit_range = 200
96 #if fit_range > np.size(counts):
97 #    fit_range = np.size(counts)
98 #if np.size(np.nonzero((counts[0:fit_range]-accidentals)<0)) >
98 #    ↪ 0:
99 #    if fit_range > np.nonzero((counts[0:fit_range]-accidentals)
99 #        ↪ <0)[0][0]:
100 #        fit_range = np.nonzero((counts[0:fit_range]-accidentals)
100 #            ↪ <0)[0][0]
```

```

101 #fid .write( '\tFit over the first {0:d} bins.\n'.format(
102     ↪ fit_range))
103
104 fit = np.polyfit(bins[fit_mask], np.log(counts[fit_mask]-
105     ↪ accidentals), 1)
106 #fit = np.polyfit(bins[0:fit_range], np.log(counts[0:fit_range]
107     ↪ ]-accidentals), 1)
108 fid .write( '\t1/tau: {0:f} us \tR: {1:f}\n'.format(-1.0/fit[0],
109     ↪ exp(fit[1])))
110 fid .write( '\n')
111 return accidentals, -1.0/fit[0], exp(fit[1])
112
113 # RossiA histograms capture data from event_list.
114 # Takes event_list (must have [ 'time' ] column) and histograms
115     ↪ with
116 #     MAX_TIME window. Prints data to file (fid).
117 # Returns histogram bins and counts.
118 def RossiA(event_list, MAX_TIME, bin_width, fid):
119     fid .write( 'RossiA...\n')
120     #MAX_TIME = 50 # microseconds
121     #bin_width = 0.2 # microseconds
122     fid .write( '\tMaximum time: {0:f} us\n'.format(MAX_TIME))
123     fid .write( '\tBin width: {0:f} us\n'.format(bin_width))
124
125 bins = np.arange(0, MAX_TIME, bin_width)
126 counts = np.zeros((np.size(bins)))
127
128 MAX_EVENTS = np.size(event_list[ 'time' ])
129 for i, time_i in enumerate(event_list[ 'time' ]):
130     j=i+1
131     time = event_list[ 'time' ][j]
132
133     if (time+MAX_TIME*1.0e-6 > event_list[ 'time' ][MAX_EVENTS-1]
134         or j > MAX_EVENTS):
135         break
136
137     while (time < time_i+MAX_TIME*1.0e-6 and j < MAX_EVENTS):
138         bin_num = floor((time - time_i)*1.0e6/bin_width)
139         counts[bin_num]+=1
140
141         j+=1
142         time = event_list[ 'time' ][j]
143
144 fid .write( '\tRossi alpha histogram\n')

```

```

141     for i, bin_val in enumerate(bins):
142         fid.write( '\t\t{0:04f}\t{1:f}\n'.format(bin_val, counts[i]) )
143         fid.write( '\n' )
144     return bins, counts
145
146 # FeynmanY histograms event_list capture data.
147 # Takes event_list (must have [ 'time' ] column) and inverse
148 #   ↪ neutron
149 #   lifetime (s). Prints to file (fid).
150 # Returns Feynman-Y value.
151 def FeynmanY(event_list, beta, gate_width_sweep, tau, S_0, F_0,
152   ↪ NSS, psi, nu_bar_0, nu_bar_1, D_0, D_1, fid):
153     fid.write( 'FeynmanY...\n' )
154     fid.write( '\tY_m = (C^2_bar/C_bar) - C_bar - 1 \n' )
155     fid.write( '\tY_c = (Y_m + 2*tau*(C_bar/T_0))/(1 - 4*tau*h) \n\
156   ↪ n' )
157
158 MAXMULT = 1000
159 feynman_y = np.zeros((np.size(gate_width_sweep),2))
160 fy_mult = np.zeros((np.size(gate_width_sweep),MAXMULT))
161
162 gate_num = 0
163 prev_Y = 0
164 for i, gate_width in enumerate(gate_width_sweep):
165     counts = np.zeros((MAXMULT))
166     current_time = gate_width
167     end_time = event_list[ 'time' ][ np.size(event_list[ 'time' ]) - 1 ]
168
169     current_event = 0
170     hit_count = 0
171     while current_time < end_time:
172         if event_list[ 'time' ][ current_event ] < current_time:
173             hit_count+=1
174             current_event+=1
175             # do dead-time stuff...
176         else:
177             current_time+=gate_width
178             counts[ hit_count ]+=1
179             hit_count = 0
180
181     fy_mult[ i ] = counts
182     mean = (np.sum(np.multiply(counts,np.arange(0,MAXMULT,1)))/
183   ↪ np.sum(counts))
184     variance = ((np.sum(np.multiply(counts,np.square(np.arange(
185   ↪ (0,MAXMULT,1)))))
```

```

181         / np.sum(counts)) - mean**2)
182
183     mean_c2 = np.sum(np.multiply(counts, np.square(np.arange(0,
184         ↪ MAXMULT, 1)))) / np.sum(counts)
185     mean_c3 = np.sum(np.multiply(counts, np.power(np.arange(0,
186         ↪ MAXMULT, 1), 3))) / np.sum(counts)
187     mean_c4 = np.sum(np.multiply(counts, np.power(np.arange(0,
188         ↪ MAXMULT, 1), 4))) / np.sum(counts)
189
190
191     mean_var = (mean_c2 - mean**2) / (np.sum(counts) - 1)
192     mean_c2_var = (mean_c4 - mean_c2**2) / (np.sum(counts) - 1)
193     cc_std = (mean_c3 - mean_c2 * mean) / (np.sum(counts) - 1)
194
195     feynman_y[gate_num, 1] = ((mean_c2 / mean**2 + 1)**2 * mean_var
196         + (1 / mean)**2 * mean_c2_var
197         - 2 * (mean_c2 / mean**3 + 1 / mean) * cc_std)
198     #feynman_y[gate_num, 1] = ( ((np.sum(np.multiply(counts, np.
199         ↪ square(np.arange(0, MAXMULT, 1)))) / np.sum(counts)) / mean)**2
200         #           *((mean_var / mean**2 + mean_c2_var / ((np.sum(np.
201             ↪ . multiply(counts, np.square(np.arange(0, MAXMULT, 1)))) / np.
202             ↪ sum(counts)) / mean)**2))
203         #           + mean_var)**0.5
204
205     feynman_y[gate_num, 0] = variance / mean - 1 # Y_m
206
207     fid.write('\tT_0: {0:f}\n'.format(gate_width))
208     fid.write('\tCounts: ')
209     for i in range(0, 10):
210         fid.write('\t{0:.0f}, '.format(counts[i]))
211     fid.write('\n')
212     fid.write('\t\tY_m = {1:f} Y_m/Y_m-1 = {2:f}'.format(
213         gate_width, variance / mean - 1, (variance / mean - 1) / prev_Y))
214     fid.write('\t\tC_bar: {0:f} var: {1:f}\n'.format(mean,
215         variance))
216     prev_Y = feynman_y[gate_num - 1, 0]
217
218     # FindMTYC(Y_m, T_0, C_bar, beta) for M_T(Y_c=Y_m), h(M_T),
219     ↪ Y_c(h), M.T(Y_c)... until small change in M_T
220     FindMTYC(feynman_y[gate_num, 0], feynman_y[gate_num, 1],
221         ↪ gate_width,
222             mean, beta, tau,
223             S_0, F_0, NSS,
224             psi, nu_bar_0, nu_bar_1,
225             D_0, D_1, fid)
226     gate_num += 1

```

```

216     fid . write( '\n' )
217
218     fid . write( '\n' )
219     return feynman_y , fy_mult
220
221 # ApplyTimeDist time distributes MCNP capture events .
222 # Takes event_list and uses specified activity hist_rate to
223 #   ↪ distribute
224 #   capture events (time indp histories) .
225 # Returns modified event_list .
226 def ApplyTimeDist(event_list , hist_rate , fid ):
227     fid . write( 'ApplyTimeDist... \n' )
228     current_time = 0
229     current_hist = -1
230     total_hist = event_list [ 'hist' ][ np.size( event_list [ 'hist' ]) -1 ]
231
232     fid . write( '\tMean time between events: {0:2.4e} s\n' .format
233                 ↪ (1.0 / hist_rate ))
234     fid . write( '\tActivity: {0:f} nps/s\n' .format( hist_rate ))
235     time_list = np.random.exponential( scale=1.0 / hist_rate , size=(
236                 ↪ int( total_hist ) , 1 ))
237     time_list = np.cumsum( time_list )
238
239     for i , hist in enumerate( event_list [ 'hist' ]):
240         event_list [ 'time' ][ i ] = time_list [ hist -1 ] + event_list [ 'time'
241                         ↪ ][ i ]
242
243     event_list = np.sort( event_list , axis=0 , order='time' )
244     fid . write( '\tTotal time: {0:f}\n' .format( float( time_list [
245                 ↪ total_hist -1 ])))
246
247     fid . write( '\n' )
248     return event_list
249
250 # ApplyDeadTime applies specified dead time to event_list
251 # Takes event_list and uses specified dead time to modify
252 #   ↪ event_list
253 # Returns event_list
254 def ApplyDeadTime(event_list , tau , fid ):
255     fid . write( 'ApplyDeadTime... \n' )
256     mask = np.ones( len( event_list ) , dtype=bool )
257
258     for i in range(1 , len( mask )):
259         prior_event = i -1

```

```

254     while (event_list['time'][prior_event]+tau > event_list['
255         ↪ time'][i]
256             and prior_event >= 0):
257                 if (event_list['cell'][prior_event] == event_list['cell'][
258                     ↪ i]):
259                         mask[i] = False
260
261         prior_event -= 1
262
263 fid.write('\tDead time losses: {0:d}\n'.format(len(event_list
264         ↪ )-np.sum(mask)))
265 fid.write('\n')
266 event_list = event_list[mask]
267 return event_list
268
269 # GetCapEvents loads events from ptrac file
270 # Takes ptrack file path (ptrac_file) and loads He-3 captures in
271 #     file to event_list ('hist','cell','time').
272 # Returns event_list
273 def GetCapEvents(ptrac_file, fid):
274     fid.write('GetCapEvents...\n')
275     fid.write('\tReading file: {0:s}\n'.format(ptrac_file))
276
277     if (ptrac_file[-3:] == '.h5'):
278         ptrac_obj = Ptrac(ptrac_file,
279                           Ptrac.HDF5_PTRAC)
280     else:
281         ptrac_obj = Ptrac(ptrac_file,
282                           Ptrac.BIN_PTRAC)
283
284     hists = ptrac_obj.ReadHistories(100)
285     counts = np.zeros((100,1))
286     count = 0
287     total_count = 0
288     hist_num = 0
289     gross_count = 0
290
291     MAX_EVENTS = int(50000000)
292     reach_max = False
293     cap_type = np.dtype([( 'hist', np.uint32 ),( 'cell', np.int16 ),( 'time', np.float64 )])
294     event_list = np.zeros((MAX_EVENTS,1), dtype=cap_type)
295
296     while hists:
297         if reach_max:

```

```

295     break
296 else:
297     for hist in hists:
298         ptrac_nps = hist.GetNPS()
299         if reach_max:
300             break
301         else:
302             for i in range(hist.GetNumEvents()):
303                 event = hist.GetEvent(i)
304                 if total_count < MAX_EVENTS:
305                     gross_count +=1
306                     if event.Get(Ptrac.TERMINATION_TYPE)== 12:
307                         event_list[ 'cell' ][ total_count ] = event.Get(
308                             ↪ Ptrac.CELL)
309                         event_list[ 'time' ][ total_count ] = event.Get(
310                             ↪ Ptrac.TIME)*1.0e-9
311                         event_list[ 'hist' ][ total_count ] = ptrac_nps.NPS
312                         ↪ ()
313                         count+=1
314                         total_count+=1
315                     else:
316                         fid .write( '\tWARNING: Reached maximum number of
317                         ↪ events {0:d}.\n'.format(MAX_EVENTS) )
318                         reach_max = True
319                         break
320
321             counts[ count ]+=1
322             count = 0
323             hist_num+=1
324
325             hists = ptrac_obj .ReadHistories(10000)
326
327             fid .write( '\tLast NPS: {0:d}\n'.format(int(event_list[ 'hist' ][
328                             ↪ total_count -1])) )
329             fid .write( '\tTime independent multiplets.\n' )
330             for i in range(0,20):
331                 fid .write( '\t{0:02d}: \t\t{1:f}\n'.format(i , counts[i,0]) )
332                 fid .write( '\tTotal counts: {0:d}\n'.format(total_count) )
333                 fid .write( '\tFraction of useful ptrac events: {0:f}\n'.format(
334                             ↪ float(total_count)/gross_count) )
335
336             fid .write( '\n' )
337             return event_list[0:total_count]
338
339 def main():

```

```

334 # Read the configuration file and pull out relevant parameters
335     ↪ .
336 # General script parameters here.
337 config_file = 'config.ini'
338 if (len(sys.argv) > 1):
339     config_file = str(sys.argv[1])
340
341 config = configparser.ConfigParser()
342 config.read(config_file)
343
344 file_name = config['general']['ptrac_file']
345 out_file_extension = config['general']['output_ext']
346 hist_rate = float(config['general']['hist_rate'])
347
348 out_file_name = 'FY_analysis.{ext}.txt'.format(ext=
349     ↪ out_file_extension)
350
351 create_event_list = (config['general']['create_event_list'] ==
352     ↪ 'True')
353 event_list_path = config['general']['event_list_path']
354 multiplicity_analysis = (config['general'][
355     ↪ 'multiplicity_analysis'] == 'True')
356
357 # Multiplicity analysis parameters here.
358 rossi_max_time = float(config['system']['rossi_max_time'])
359 rossi_bin_step = float(config['system']['rossi_bin_step'])
360 rossi_accidental_start = float(config['system'][
361     ↪ 'rossi_accidental_start'])
362 rossi_fit_start = float(config['system']['rossi_fit_start'])
363 rossi_fit_end = float(config['system']['rossi_fit_end'])
364 rossi_bin_step = float(config['system']['rossi_bin_step'])
365
366 tau = float(config['system']['tau'])
367 S_0 = float(config['system']['S_0'])
368 F_0 = float(config['system']['F_0'])
369 gate_width_sweep = np.arange( float(config['system'][
370     ↪ 'sweep_start']),
371                                     float(config['system'][[
372     ↪ 'sweep_end'],
373                                     float(config['system'][[
374     ↪ 'sweep_step']])))
374 NSS = float(config['system']['NSS'])
375 nu_bar_0 = float(config['system']['nu_bar_0'])
376 nu_bar_1 = float(config['system']['nu_bar_1'])
377 D_0 = float(config['system']['D_0'])

```

```

371 D_1 = float(config[ 'system' ][ 'D_1' ])
372 psi = float(config[ 'system' ][ 'psi' ])
373
374 # Call the reading, processing, and analysis functions with
375 # → output
376 # to a simple text file and various NumPy files.
377 with open(out_file_name, 'w') as fid:
378     if create_event_list:
379         event_list = GetCapEvents(file_name, fid)
380         np.save('event_list.{ext}'.format(ext=
381             ↪ out_file_extension), event_list)
382         event_list = ApplyTimeDist(event_list, hist_rate, fid)
383         np.save('event_list_td.{ext}'.format(ext=
384             ↪ out_file_extension), event_list)
385         event_list = ApplyDeadTime(event_list, tau, fid)
386         np.save('event_list_td_dt.{ext}'.format(ext=
387             ↪ out_file_extension), event_list)
388     else:
389         event_list = np.load(event_list_path)
390
391 with open(out_file_name, 'a') as fid:
392     if multiplicity_analysis:
393         # Rossi-alpha
394         [rossi_bins, rossi] = RossiA( event_list,
395                                         rossi_max_time,
396                                         rossi_bin_step,
397                                         fid)
398         np.save('rossi_alpha.{ext}'.format(ext=
399             ↪ out_file_extension), rossi)
400         np.save('rossi_alpha_bins.{ext}'.format(ext=
401             ↪ out_file_extension), rossi_bins)
402
403         fit = FitRossiA( rossi_bins, rossi,
404                           rossi_accidental_start,
405                           rossi_fit_start,
406                           rossi_fit_end,
407                           fid)
408         np.save('rossi_fit.{ext}'.format(ext=
409             ↪ out_file_extension), fit)
410         beta = 1.0 / fit[1]*1e6
411
412         # Feynman-Y
413         feynman_y, fy_mult = FeynmanY( event_list, beta,
414             ↪ gate_width_sweep, tau, S_0, F_0, NSS, psi, nu_bar_0,
415             ↪ nu_bar_1, D_0, D_1, fid)

```

```

407     fits = FeynmanBetaFit(feynman_y[:,0], gate_width_sweep, tau,
408     ↪ fid)
409     np.save('feynman_gates.{ ext }.npy'.format(ext=
410     ↪ out_file_extension), gate_width_sweep)
411     np.save('feynman_y_fit.{ ext }.npy'.format(ext=
412     ↪ out_file_extension), fits)
413     np.save('feynman_y.{ ext }.npy'.format(ext=
414     ↪ out_file_extension), feynman_y)
415     np.save('fy_mult.{ ext }.npy'.format(ext=out_file_extension)
416     ↪ , fy_mult)
417
418 if __name__ == '__main__':
419     main()

```

### 0.3 MCNP6 model: Cf-252 point source

The MCNP model describes a neutron measurement with the Fission Meter of a point Cf-252 spontaneous fission source.

```
1 Ortec Fission Meter 7.5 atm
2 c based on brochure info
3 c
4 c
5 c
6 c          Cells
7 c
8 c
9 c          Fission Meter Cells
10 c
11 c
12 c He-3 tubes - right side detector bank
13 1 3 -0.0009358 25 -26 -1
14 2 3 -0.0009358 25 -26 -2
15 3 3 -0.0009358 25 -26 -3
16 4 3 -0.0009358 25 -26 -4
17 5 3 -0.0009358 25 -26 -5
18 6 3 -0.0009358 25 -26 -6
19 7 3 -0.0009358 25 -26 -7
20 8 3 -0.0009358 25 -26 -8
21 9 3 -0.0009358 25 -26 -9
22 10 3 -0.0009358 25 -26 -10
23 11 3 -0.0009358 25 -26 -11
24 12 3 -0.0009358 25 -26 -12
25 13 3 -0.0009358 25 -26 -13
26 14 3 -0.0009358 25 -26 -14
27 15 3 -0.0009358 25 -26 -15
28 c He-3 tubes - left side detector bank
29 16 3 -0.0009358 25 -26 -28
30 17 3 -0.0009358 25 -26 -29
31 18 3 -0.0009358 25 -26 -30
32 19 3 -0.0009358 25 -26 -31
33 20 3 -0.0009358 25 -26 -32
34 21 3 -0.0009358 25 -26 -33
35 22 3 -0.0009358 25 -26 -34
36 23 3 -0.0009358 25 -26 -35
```

```

37 24 3 -0.0009358    25 -26 -36
38 25 3 -0.0009358    25 -26 -37
39 26 3 -0.0009358    25 -26 -38
40 27 3 -0.0009358    25 -26 -39
41 28 3 -0.0009358    25 -26 -40
42 29 3 -0.0009358    25 -26 -41
43 30 3 -0.0009358    25 -26 -42
44 c moderator - right
45 40 4 -0.96    -17 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
46 c air - right
47 41 6 -0.0012    (-18 17 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15):
48           (-18 17 -25):(-18 17 26)
49 c Aluminum case - right
50 42 5 -2.6989 18 -19
51 c moderator - left
52 50 4 -0.96    -44 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42
53 c air - left
54 51 6 -0.0012    (-45 44 28 29 30 31 32 33 34 35 36 37 38 39 40
55           ↪ 41 42):(-45 44 -25):(-45 44 26)
56 c Aluminum case - left
57 52 5 -2.6989 45 -46
58 c
59 c pretendium
60 100      252 -15.1    -100          $ Cf-252
61 500      6   -0.001205    19 46 100 -99          $ inside room
62 c
63 99 0           99
64
65 c
66 c
67 c           surface descriptions
68 c
69 c
70 c
71 c Right side detector bank
72 1 1 c/z 3.81 0.0 1.26997
73 2 1 c/z 6.35 0.0 1.26997
74 3 1 c/z 8.89 0.0 1.26997
75 4 1 c/z 11.43 0.0 1.26997
76 5 1 c/z 13.97 0.0 1.26997

```

```

77 6 1 c/z 16.51 0.0 1.26997
78 7 1 c/z 19.05 0.0 1.26997
79 8 1 c/z 21.59 0.0 1.26997
80 9 1 c/z 24.13 0.0 1.26997
81 10 1 c/z 26.67 0.0 1.26997
82 11 1 c/z 29.21 0.0 1.26997
83 12 1 c/z 31.75 0.0 1.26997
84 13 1 c/z 34.29 0.0 1.26997
85 14 1 c/z 36.83 0.0 1.26997
86 15 1 c/z 39.37 0.0 1.26997
87 c Right side moderator
88 17 1 rpp 0.2 43.18 -3.62 -0.52 -24.13 24.13
89 18 1 rpp 0.1 43.28 -3.72 1.37 -24.23 24.23
90 19 1 rpp 0 43.38 -3.82 1.47 -24.33 24.33
91 c Z planes for both banks
92 25 pz -24.13
93 26 pz 24.13
94 c Left side detector bank
95 28 2 c/z -3.81 0.0 1.26997
96 29 2 c/z -6.35 0.0 1.26997
97 30 2 c/z -8.89 0.0 1.26997
98 31 2 c/z -11.43 0.0 1.26997
99 32 2 c/z -13.97 0.0 1.26997
100 33 2 c/z -16.51 0.0 1.26997
101 34 2 c/z -19.05 0.0 1.26997
102 35 2 c/z -21.59 0.0 1.26997
103 36 2 c/z -24.13 0.0 1.26997
104 37 2 c/z -26.67 0.0 1.26997
105 38 2 c/z -29.21 0.0 1.26997
106 39 2 c/z -31.75 0.0 1.26997
107 40 2 c/z -34.29 0.0 1.26997
108 41 2 c/z -36.83 0.0 1.26997
109 42 2 c/z -39.37 0.0 1.26997
110 c Left side moderator
111 44 2 rpp -43.18 -0.2 -3.62 -0.52 -24.13 24.13
112 45 2 rpp -43.28 -0.1 -3.72 1.37 -24.23 24.23
113 46 2 rpp -43.38 0.0 -3.82 1.47 -24.33 24.33
114 c Cf
115 100 3 so 0.0000001
116 99 so 2146.8503 $
117
118 c
119 c
    ↪ ~~~~~
    ↪ ~~~~~
```

```

120 c          Data cards
121 c
122 c
123 mode n      $ neutron mode
124 imp:n 1 37r 0
125 c
126 c
127 prdmp j 1e6 1$
128 nps 1e8 $
129 CUT:N 2J 0 0$
130 PHYS:N J 100 3J $
131 totnu$$
132 print
133 c
134 c      Transformation for right bank
135 *tr1  1.5 -47.0 0   45 45 90 135 45 90 90 90 0
136 c      Transformation for left bank
137 *tr2  -1.5 -47.0 0   45 135 90 45 45 90 90 90 0
138 c      Translation for object
139 TR3 0 0 0
140 c
141 c
142 c      ..... source specifications .....
143 c      The ptrac card is set to record all events that terminate in
144 c      cells
145 c      1 through 30. The flushnps card is necessary for HDF5
146 c      writing
147 c      to avoid high memory usage in storing events.
148 c      Ptrac card for MCNP6-devel with HDF5
149 PTRAC EVENT=TER FILTER=1,30,cel $
150     FLUSHNPS=1e4 FILE=HDF5$
151 c      Ptrac card for MCNP6.2.1
152 c      PTRAC EVENT=CAP WRITE=ALL MAX=1e6
153 SDEF PAR=SF POS= 0 0 0 CEL=100 RAD=D2
154 SI2 0 0.0000001
155 c
156 c
157 c
158 c      Tally
159 f8:n (1 2 3 4 5 6 7 8 9 10
160       11 12 13 14 15 16 17 18 19 20

```

```

161      21 22 23 24 25 26 27 28 29 30)
162 ft8      CAP 2003 gate 100 400
163 c
164 c
165 c
    ↪ ~~~~~
    ↪ ~~~~~
166 c Materials for Fission Meter
167 c
    ↪ ~~~~~
    ↪ ~~~~~
168 c
169 c —— He-3 ———
170 m3      2003   1.0      $ He-3
171 c
172 c —— Polythene ———
173 m4      6000 -0.856      $ C
174          1001 -0.144      $ H
175 mt4     poly.60t      $ S(a,B) card
176 c
177 c —— Al ———
178 m5      13027  1.0      $ Al
179 c
180 c —— Air [from PNNL compendium of material composition (density
    ↪ =0.001205g/cc]
181 m6      6000 -0.000214      $ C
182          7014 -0.755268      $ N
183          8016 -0.231781      $ O
184          18000 -0.012827      $ Ar
185 c
186 c —— HDPE ———
187 m7      1001  0.666662
188          6000  0.333338
189 c
190 c —— Pb for pretendium -
191 m8      82000  1.0
192          98252 1e-11
193 c
194 c —— 1040 steel ———
195 m9      6000  0.0040
196          15031 0.0004
197          16000 0.0005
198          25055 0.0075
199          26000 0.9876
200 c

```

```
201 c — Cf-252 for source —  
202 m252      98252 100      $ Californium (100% Cf252) - d=15.1g/cc  
203   ↳      for neutron point source $  
204 c
```

## 0.4 Python script configuration: Cf-252 point source

The configuration file for the ptrac\_feynman\_y.py script describes a neutron measurement with the Fission Meter of a point Cf-252 spontaneous fission source.

```
1 # General script parameters
2 [general]
3
4 ptrac_file = fm_cf_mcnp.1e8.out.p.h5
5
6 output_ext = 1e8.out
7
8 # History activity in s^-1
9 hist_rate = 1e5
10
11 # True or False: read ptrac, time distribute, apply dead time,
12 #                   write
12 create_event_list = True
13
14 # Event list path to read; if you already have one and want to
14 #                   skip
15 #       to multiplicity analysis.
16 event_list_path = event_list_td_dt.1e8.test.npy
17
18 multiplicity_analysis = True
19
20 # system and material parameters
21 [system]
22
23 # Uncorrelated neutron rate in [n/s] i.e. (alpha,n)
24 S_0 = 0
25
26 # Rossi alpha histogram range [us]
27 rossi_max_time = 100
28 rossi_bin_step = 1
29 rossi_accidental_start = 75
30 rossi_fit_start = 5
31 rossi_fit_end = 50
32
33 # Feynman-Y gate widths [s]
34 sweep_start = 1e-6
35 sweep_end = 256e-6
```

```

36 sweep_step = 4e-6
37
38 # Detector dead time [s]
39 tau = 3.125e-08
40
41 # Starter fission rate in [f/s]
42 F_0 = 3.75e5
43
44 # Total neutron source strength [n/s]
45 NSS = 376727.51
46
47 # nu_bar in starter material [n/f]
48 nu_bar_0 = 3.76
49
50 # nu_bar in multiplying material [n/f]
51 nu_bar_1 = 4.415
52
53 # Diven-number for starter material
54 D_0 = 3.23
55
56 # Diven-number for multiplying material
57 D_1 = 3.23
58
59 # Transmission ratio
60 psi = 1

```

## 0.5 MCNP6 model: TAI 1

The MCNP model describes a neutron measurement with the Fission Meter of the TAI 1 source.

```

1 Ortec Fission Meter 7.5 atm
2 c based on brochure info
3 c
4 c
5 c
6 c           Cells
7 c
8 c
9 c   Fission Meter Cells
10 c
11 c
12 c He-3 tubes - right side detector bank
13 1  3 -0.0009358  25 -26 -1
14 2  3 -0.0009358  25 -26 -2
15 3  3 -0.0009358  25 -26 -3
16 4  3 -0.0009358  25 -26 -4
17 5  3 -0.0009358  25 -26 -5
18 6  3 -0.0009358  25 -26 -6
19 7  3 -0.0009358  25 -26 -7
20 8  3 -0.0009358  25 -26 -8
21 9  3 -0.0009358  25 -26 -9
22 10 3 -0.0009358  25 -26 -10
23 11 3 -0.0009358  25 -26 -11
24 12 3 -0.0009358  25 -26 -12
25 13 3 -0.0009358  25 -26 -13
26 14 3 -0.0009358  25 -26 -14
27 15 3 -0.0009358  25 -26 -15
28 c He-3 tubes - left side detector bank
29 16 3 -0.0009358  25 -26 -28
30 17 3 -0.0009358  25 -26 -29
31 18 3 -0.0009358  25 -26 -30
32 19 3 -0.0009358  25 -26 -31
33 20 3 -0.0009358  25 -26 -32
34 21 3 -0.0009358  25 -26 -33
35 22 3 -0.0009358  25 -26 -34
36 23 3 -0.0009358  25 -26 -35

```

```

37 24 3 -0.0009358    25 -26 -36
38 25 3 -0.0009358    25 -26 -37
39 26 3 -0.0009358    25 -26 -38
40 27 3 -0.0009358    25 -26 -39
41 28 3 -0.0009358    25 -26 -40
42 29 3 -0.0009358    25 -26 -41
43 30 3 -0.0009358    25 -26 -42
44 c moderator - right
45 40 4 -0.96    -17 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
46 c air - right
47 41 6 -0.0012    (-18 17 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15):
48           (-18 17 -25):(-18 17 26)
49 c Aluminum case - right
50 42 5 -2.6989 18 -19
51 c moderator - left
52 50 4 -0.96    -44 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42
53 c air - left
54 51 6 -0.0012    (-45 44 28 29 30 31 32 33 34 35 36 37 38 39 40
55           ↪ 41 42):(-45 44 -25):(-45 44 26)
56 c Aluminum case - left
57 52 5 -2.6989 45 -46
58 c
59 c pretendium
60 100      8 -19.35    -100          $ pretendium upper
61 101      8 -19.35    -110          $ pretendium lower
62 200      7 -0.95     100 110 -200      $ HDPE
63 300      6 -0.001205 200 -300      $ air in barrel
64 301      9 -7.845    300 -310        $ barrel
65 c
66 500      6 -0.001205   19 46 310 -99      $ inside room
67 c
68 99 0          99
69
70 c
71 c
72 c           ↪ ~~~~~
73 c           ↪ ~~~~~
74 c           ↪ ~~~~~
75 c
76 c Right side detector bank

```

```

77 1 1 c/z 3.81 0.0 1.26997
78 2 1 c/z 6.35 0.0 1.26997
79 3 1 c/z 8.89 0.0 1.26997
80 4 1 c/z 11.43 0.0 1.26997
81 5 1 c/z 13.97 0.0 1.26997
82 6 1 c/z 16.51 0.0 1.26997
83 7 1 c/z 19.05 0.0 1.26997
84 8 1 c/z 21.59 0.0 1.26997
85 9 1 c/z 24.13 0.0 1.26997
86 10 1 c/z 26.67 0.0 1.26997
87 11 1 c/z 29.21 0.0 1.26997
88 12 1 c/z 31.75 0.0 1.26997
89 13 1 c/z 34.29 0.0 1.26997
90 14 1 c/z 36.83 0.0 1.26997
91 15 1 c/z 39.37 0.0 1.26997
92 c Right side moderator
93 17 1 rpp 0.2 43.18 -3.62 -0.52 -24.13 24.13
94 18 1 rpp 0.1 43.28 -3.72 1.37 -24.23 24.23
95 19 1 rpp 0 43.38 -3.82 1.47 -24.33 24.33
96 c Z planes for both banks
97 25 pz -24.13
98 26 pz 24.13
99 c Left side detector bank
100 28 2 c/z -3.81 0.0 1.26997
101 29 2 c/z -6.35 0.0 1.26997
102 30 2 c/z -8.89 0.0 1.26997
103 31 2 c/z -11.43 0.0 1.26997
104 32 2 c/z -13.97 0.0 1.26997
105 33 2 c/z -16.51 0.0 1.26997
106 34 2 c/z -19.05 0.0 1.26997
107 35 2 c/z -21.59 0.0 1.26997
108 36 2 c/z -24.13 0.0 1.26997
109 37 2 c/z -26.67 0.0 1.26997
110 38 2 c/z -29.21 0.0 1.26997
111 39 2 c/z -31.75 0.0 1.26997
112 40 2 c/z -34.29 0.0 1.26997
113 41 2 c/z -36.83 0.0 1.26997
114 42 2 c/z -39.37 0.0 1.26997
115 c Left side moderator
116 44 2 rpp -43.18 -0.2 -3.62 -0.52 -24.13 24.13
117 45 2 rpp -43.28 -0.1 -3.72 1.37 -24.23 24.23
118 46 2 rpp -43.38 0.0 -3.82 1.47 -24.33 24.33
119 c pretendum
120 100 3 rpp -8.85 8.85 -8.85 8.85 1.5 3.5
121 110 3 rpp -8.85 8.85 -8.85 8.85 -8.5 -6.5

```

```

122 200 3 rcc 0 0 -12.5 0 0 25 15
123 300 3 rcc 0 0 -42.55 0 0 85.1 28.6
124 310 3 rcc 0 0 -43.185 0 0 86.37 29.235
125 99 so 2146.8503 $
126
127 c
128 c
    ↪ ~~~~~
    ↪ ~~~~~
129 c          Data cards
130 c
    ↪ ~~~~~
    ↪ ~~~~~
131 c
132 mode n      $ neutron mode
133 imp:n 1 41r 0
134 c
135 c
136 prdmp j 1e6 1$
137 nps 1e8 $
138 CUT:N 2J 0 0$
139 PHYS:N J 100 3J $
140 totnu$$
141 print
142 c
143 c      Transformation for right bank
144 *tr1 1.5 -47.0 0 45 45 90 135 45 90 90 90 0
145 c      Transformation for left bank
146 *tr2 -1.5 -47.0 0 45 135 90 45 45 90 90 90 0
147 c      Translation for object
148 TR3 0 0 0
149 c
150 c
151 c      ..... source specifications .....
152 c      The ptrac card is set to record all events that terminate in
    ↪ cells
153 c      1 through 30. The flushnps card is necessary for HDF5
    ↪ writing
154 c      to avoid high memory usage in storing events.
155 c
156 c      Ptrac card for MCNP6-devel with HDF5
157 PTRAC EVENT=TER FILTER=1,30,cel $
158     FLUSHNPS=1e4 FILE=HDF5$
159 c      Ptrac card for MCNP6.2.1
160 c      PTRAC EVENT=CAP WRITE=ALL MAX=1e6

```

```

161 c
162 SDEF PAR=SF CEL=d1 Z=FCEL=d2 X=d5 Y=d6
163 SI1 L 100 101
164 SP1 1 1
165 DS2 S 3 4
166 SI4 -8.5 -6.5
167 SP4 0 1
168 SI3 1.5 3.5
169 SP3 0 1
170 SI5 -8.85 8.85
171 SP5 0 1
172 SI6 -8.85 8.85
173 SP6 0 1
174 c
175 c
176 c
177 c Tally
178 f8:n (1 2 3 4 5 6 7 8 9 10
179 11 12 13 14 15 16 17 18 19 20
180 21 22 23 24 25 26 27 28 29 30)
181 ft8 CAP 2003 gate 100 400
182 c
183 c
184 c
    ↪ ~~~~~
    ↪ ~~~~~
185 c Materials for Fission Meter
186 c
    ↪ ~~~~~
    ↪ ~~~~~
187 c
188 c —— He-3 —————
189 m3 2003 1.0 $ He-3
190 c
191 c —— Polythene —————
192 m4 6000 -0.856 $ C
193 1001 -0.144 $ H
194 mt4 poly.60 t $ S(a,B) card
195 c
196 c —— Al —————
197 m5 13027 1.0 $ Al
198 c
199 c —— Air [from PNNL compendium of material composition (density
    ↪ =0.001205g/cc]
200 m6 6000 -0.000214 $ C

```

```

201      7014 -0.755268 $ N
202      8016 -0.231781 $ O
203      18000 -0.012827 $ Ar
204 c
205 c ---- HDPE -----
206 m7     1001  0.666662
207          6000  0.333338
208 c
209 c ---- Pb for pretendium -
210 m8     82000 1.0
211          98252 1e-11
212 c
213 c ---- 1040 steel -----
214 m9     6000  0.0040
215          15031 0.0004
216          16000 0.0005
217          25055 0.0075
218          26000 0.9876
219 c
220 c ---- Cf-252 for source --
221 m252    98252 100      $ Californium (100% Cf252) - d=15.1g/cc
222          ↳ for neutron point source $
223 c

```

## 0.6 Python script configuration: TAI 1

The configuration file for the ptrac\_feynman\_y.py script describes a neutron measurement with the Fission Meter of the TAI 1 source.

```
1 # General script parameters
2 [general]
3
4 ptrac_file = fm_tai_mcnp.1e8.out.p.h5
5
6 output_ext = 1e8
7
8 # History activity in s^-1
9 hist_rate = 6.45455e5
10
11 # True or False: read ptrac, time distribute, apply dead time,
12 # write
12 create_event_list = True
13
14 # Event list path to read; if you already have one and want to
14 # skip
15 # to multiplicity analysis.
16 event_list_path = event_list_td_dt.1e8.npy
17
18 multiplicity_analysis = True
19
20 # system and material parameters
21 [system]
22
23 # Uncorrelated neutron rate in [n/s] i.e. (alpha,n)
24 S_0 = 0
25
26 # Rossi alpha histogram range [us]
27 rossi_max_time = 100
28 rossi_bin_step = 1
29 rossi_accidental_start = 75
30 rossi_fit_start = 5
31 rossi_fit_end = 50
32
33 # Feynman-Y gate widths [s]
34 sweep_start = 30e-6
35 sweep_end = 400e-6
36 sweep_step = 8e-6
37
38 # Detector dead time [s]
```

```
39 tau = 3.125e-08
40
41 # Starter fission rate in [f/s]
42 F_0 = 6.45455e5
43
44 # Total neutron source strength [n/s]
45 NSS = 2424860.
46
47 # nu_bar in starter material [n/f]
48 nu_bar_0 = 3.76
49
50 # nu_bar in multiplying material [n/f]
51 nu_bar_1 = 3.76
52
53 # Diven-number for starter material
54 D_0 = 3.23
55
56 # Diven-number for multiplying material
57 D_1 = 3.23
58
59 # Transmission ratio
60 psi = 1.0
```

## 0.7 Python results plotter

The python script plots the ptrac\_feynman\_y.py Rossi-alpha and Feynman-Y outputs.

```
1 #!/usr/bin/env python3
2 #
3 # 24 December 2020 Marcath
4 #
5 import numpy as np
6 import matplotlib.pyplot as plt
7
8 file_ext = '1e8'
9
10 # FitRossiA...
11 #   Fitting data to S(t) = A + R*e^{-t/tau}.
12 # [A, tau, R]
13 rossi_fits = np.load('rossi_fit.{ext}.npy'.format(ext=file_ext))
14 print(rossi_fits)
15
16 # FeynmanBetaFit...
17 # [K, beta, dead time]
18 fy_fits = np.load('feynman_y_fit.{ext}.npy'.format(ext=file_ext))
19 print(fy_fits)
20
21 rossi = np.load('rossi_alpha.{ext}.npy'.format(ext=file_ext))
22 rossi_bins = np.load('rossi_alpha_bins.{ext}.npy'.format(ext=
23     file_ext))
24 gate_width_sweep = np.load('feynman_gates.{ext}.npy'.format(ext=
25     file_ext))
26 feynman_y = np.load('feynman_y.{ext}.npy'.format(ext=file_ext))
27 fy_mult = np.load('fy_mult.{ext}.npy'.format(ext=file_ext))
28
29 # FeynmanBetaFunc
30 # The function is used primarily for fitting.
31 def FeynmanBetaFunc(T_0, K, beta, tau):
32     return (np.multiply(np.divide(K*(T_0-tau), T_0),
33         1-np.divide((1-np.exp(-beta*(T_0-tau))), (beta*(T_0-tau))))) )
34
35 # ~~~ Rossi-alpha plot
```

```

35 fig , ax = plt.subplots()
36 ax.errorbar(rossi_bins , rossi , yerr=np.sqrt(rossi))
37 plt.plot(rossi_bins ,
38         rossi_fits[0]+rossi_fits[2]*np.exp(-1/rossi_fits[1]*
39         ↪ rossi_bins))
39 ax.tick_params(axis='both' , labelsize=12)
40 for axis in [ 'top' , 'bottom' , 'left' , 'right' ]:
41     ax.spines[axis].set_linewidth(1.5)
42 ax.minorticks_on()
43 ax.grid(linestyle='—')
44 ax.tick_params(which='major' , direction='in' , length=6, width
45         ↪ =1.5,
46             bottom=True ,
47             top=True ,
48             left=True ,
49             right=True)
49 ax.tick_params(which='minor' , direction='in' , length=2, width
50         ↪ =1.5,
51             bottom=True ,
52             top=True ,
53             left=True ,
54             right=True)
54 ax.set_position([0.15 , 0.15 , 0.8 , 0.8] , which='both')
55 ax.ticklabel_format(scilimits=[-2,2],axis='y')
56 #ax.set_yscale('log')
57 ax.set_ylabel('Counts')
58 ax.set_xlabel(r'Time [$\mu s$]')
59 plt.legend([r'fit $A+Re^{-t/\tau}$' , 'data'])
60
# ~~~ Feynman-Y Beta
61 fig , ax = plt.subplots()
62 ax.errorbar(gate_width_sweep/1.0e-6, feynman_y[:,0] ,
63         yerr=feynman_y[:,1] , marker='o' , linestyle='')
64 ax.plot(gate_width_sweep/1.0e-6,
65         FeynmanBetaFunc(gate_width_sweep , *fy_fits[0] , fy_fits[1])
66         ↪ )
67 ax.set_xlabel(r'Gate Width [$\mu s$]')
68 ax.set_ylabel('Y')
69 ax.tick_params(axis='both' , labelsize=12)
70 for axis in [ 'top' , 'bottom' , 'left' , 'right' ]:
71     ax.spines[axis].set_linewidth(1.5)
72 ax.minorticks_on()
73 ax.grid(linestyle='—')
74 ax.tick_params(which='major' , direction='in' , length=6, width
    ↪ =1.5,

```

```

75         bottom=True ,
76         top=True ,
77         left=True ,
78         right=True)
79 ax.tick_params(which='minor' , direction='in' , length=2, width
    ↪ =1.5,
80             bottom=True ,
81             top=True ,
82             left=True ,
83             right=True)
84 ax.set_position([0.15 , 0.15 , 0.8 , 0.8] , which='both')
85 plt.legend(['Feynman-Y Beta fit' , 'data'])
86
87 # ~~~ Feynman Multiplicity
88 gate_range = range(5 , 25 , 5)
89 max_mult = 20
90
91 fig , ax = plt.subplots()
92 for i in gate_range:
93     ax.plot(range(0,max_mult) , fy_mult[i , 0:max_mult] ,
94             label=r'{gate:4.0f} $\mu s$'.format(gate=gate_width_sweep[i
    ↪ ]*1e6))
95 ax.set_xlabel('Multiplet')
96 ax.set_ylabel('Counts')
97 ax.set_yscale('log')
98 plt.legend()
99 ax.tick_params(axis='both' , labelsize=12)
100 for axis in [ 'top' , 'bottom' , 'left' , 'right' ]:
101     ax.spines[axis].set_linewidth(1.5)
102 ax.minorticks_on()
103 ax.grid(linestyle='--')
104 ax.tick_params(which='major' , direction='in' , length=6, width
    ↪ =1.5,
105             bottom=True ,
106             top=True ,
107             left=True ,
108             right=True)
109 ax.tick_params(which='minor' , direction='in' , length=2, width
    ↪ =1.5,
110             bottom=True ,
111             top=True ,
112             left=True ,
113             right=True)
114 ax.set_position([0.15 , 0.15 , 0.8 , 0.8] , which='both')
115

```

```
116 plt.show()
```